Narrative-Driven Camera Control for Cinematic Replay of Computer Games

Quentin Galvane, Rémi Ronfard, Marc Christie, Nicolas Szilas

Narrative-Driven Camera Control for Cinematic Replay of Computer Games

Quentin Galvane *
INRIA / LJK, Grenoble

Rémi Ronfard
INRIA / LJK , Grenoble

Marc Christie
University of Rennes I

Nicolas Szilas
University of Geneva
TECFA-FPSE

Figure 1: *Camera behaviors for a cinematic replay: (a) CU on Frank 3/4backright screenleft and Lili center (b) CU on Lili 3/4backright screenleft and Frank center , (c) POV Frank MCU on Lili screencenter and (d) POV Lili CU on Frank screencenter*

Abstract

This paper presents a system that generates cinematic replays for dialogue-based 3D video games. The system exploits the narrative and geometric information present in these games and automatically computes camera framings and edits to build a coherent cinematic replay of the gaming session. We propose a novel importance-driven approach to cinematic replay. Rather than relying on actions performed by characters to drive the cinematography (as in idiom-based approaches), we rely on the importance of characters in the narrative. We first devise a mechanism to compute the varying importance of the characters. We then map importances of characters with different camera specifications, and propose a novel technique that (i) automatically computes camera positions satisfying given specifications, and (ii) provides smooth camera motions when transitioning between different specifications. We demonstrate the features of our system by implementing three camera behaviors (one for master shots, one for shots on the player character, and one for reverse shots). We present results obtained by interfacing our system with a full-fledged serious game (*Nothing for Dinner*) containing several hours of 3D animated content.

Keywords: camera, cinematography, cinematic replay

1 Introduction

In the past decades, the world has experienced a continuous growth of video games in terms of popularity and quality. With the evolution of technologies, gaming companies keep improving the graphic quality of the games and rely more and more on cinematics and cinematographic techniques to enhance the gaming experience. Furthermore, with the advent of multi-player games, and the possibilities of sharing players' performance and playing experiences on the web, there is a significant demand in generating relevant cinematic replays of gaming sessions. Dedicated tools have been designed to ease the creation of replays[1], either to report game experiences, or for more aesthetic considerations such as machinima.

However, a close look at these dedicated tools shows that a lot is still done manually, typically in selecting the appropriate moments, setting the cameras, and performing edits between multiple cameras. In parallel, for the last decade, researchers in computer graphics focusing on automated virtual camera control have been proposing a number of efficient techniques to automatically place and move cameras [Halper et al. 2001; Lino and Christie 2012] as well as editing algorithms to automatically or interactively edit the shots of a movie[Elson and Riedl 2007; Lino et al. 2011b].

These approaches are mostly founded on what could be referred to as "action-based" camera control, in the sense that a typical idiom is associated to each action occurring in the 3D environment (an idiom is a stereotypical way of shooting the action, either through a single shot or a sequence of shots). A film is then constructed by computing the best sequence of shots portraying a sequence of actions performed by the characters (as in [Elson and Riedl 2007; Lino et al. 2011b; Lino et al. 2011a; Markowitz et al. 2011]).

Automated cinematography techniques have rarely been adapted to the specific case of cinematic replays (with the notable exception of [Dominguez et al. 2011a]). The problem is actually challenging. Character tracking techniques such as [Halper et al. 2001; Lino and Christie 2012] would generate cinematics of low interest by creating continuous camera motions without cuts. Idiom-based techniques [Christianson et al. 1996] would typically fail due to the inability to handle complex situations and the necessity to design idioms for many different actions and situations. Finally, optimization-based approaches such as [Elson and Riedl 2007] require the manual specification of cinematographic patterns for each situation, while [Lino et al. 2011a] maps actions to shot preferences in a straightforward way.

To overcome limitations of idiom-based techniques, as well as approaches which solely rely on characters' actions, we propose a more principled approach. Based on Hitchcock's well-known rule which states that *the size of a character on the screen should be proportional to its narrative importance in the story* [Truffaut and Scott 1967; Hawkins 2005; DeLoura 2009], we propose means to compute the individual importance of each character from the replay, and map these importances with cinematographic specifications. Importance therefore serves as an novel intermediate representation which can account for more elaborate and contextual situ-

*e-mail: quentin.galvane@inria.fr

[1] see Simatography for the Sims, Warcraft movies, replay editor or Team Fortress 2 or Total war shotgun 2

ations between characters in a replay sequence, typically including intentions, significance of the characters in the whole story, as well as causal relations between events.

Unlike idiom-based techniques, this approach to cinematography is agnostic to the type of action occurring. It only requires the provision of an importance function on the characters. The mapping between the importances and the camera specifications is not related to how importance is computed, therefore providing an independent and reusable set of cinematography techniques.

Our approach comprises a preliminary stage which enables the extraction and computation of the characters' importances from a game trace (this is specific to each game engine). Our technique is then composed of three stages: (i) mapping importances with cinematographic specifications by defining camera behaviors, (ii) animating cameras by enforcing the specified behaviors, and (iii) editing the rushes computed by the cameras.

The contributions of this paper are: (i) a character importance-based approach to drive camera placements and camera edits, thereby moving a step beyond action-based and idiom-based techniques, (ii) a novel incremental technique to convert camera specifications into camera coordinates using spherical and toric surfaces, and (iii) a smooth camera animation technique that maintains the specifications as the scene is evolving and enables smooth transitions between different camera specifications.

The benefit of the system stands in its ability to effectively convey, with a realistic and dynamic cinematic style, a dialogue-based video game session through a collection of simple and dynamic camera behaviors.

2 Related work

The seminal work of [Christianson et al. 1996] introduces the declarative camera control language (DCCL) as a general framework for generating idiom-based solutions for cinematography and film editing problems. Film idioms are recipes for obtaining good cinematography and editing in a range of predefined situations [He et al. 1996], similar to cases in case-based reasoning. The principle consists in associating typical shots or sequences of shots to specific actions or motions of the characters. As a result, DCCL uses a conversation idiom for filming conversations, a fighting idiom for filming fights, etc. Each idiom has two components: a set-up (blocking) of the cameras relative to the actors, and a state machine for switching automatically between cameras, depending on scene features such as distances between characters (i.e. cut to a specified viewpoint when characters are less than 8 meters away) or film features such as current shot duration (i.e. cut to a specified viewpoint when the shot lasts more than 5 seconds).

Elson and Riedl have proposed a lightweight cinematography system called Cambot [Elson and Riedl 2007]. Cambot takes as input a script specifying the actions and characters involved in a scene, and automatically generates the blocking (deciding where the scene should take place in the 3D environment), the staging (where the characters are placed), the cinematography and the editing. The tool relies on a two stage process, that first checks which areas are geometrically valid for the blocking and the staging. The tool then relies on a dynamic programming approach to select the optimal sequence of camera shots.

With their Darshak system, Jhala and Young propose a AI-based approach to virtual cinematography [Jhala 2006] that relies on a hierarchical partial order planner. Taking as input a structured representation of the sequence of actions in the scene, the system searches for the best idioms and best shots to convey the actions.

The specific problem of automatically generating cinematic highlights for game sessions has been addressed in [Cheong et al. 2008],[Dominguez et al. 2011b] and [Dominguez et al. 2011a]. The authors propose Afterthought, a system that analyses actions performed by the players to recognize narrative patterns expressed as Finite State Machines. The detected narrative patterns are then paired with cinematographic instructions to generate a meaningful cinematic sequence. Cinematographic instructions then trigger camera scripts in the rendering environment to compute viewpoints, taking care of visibility and optimal distances to characters. Interestingly, the proposed approach relies on patterns of actions to build the cinematographic instructions, in a principled and context-sensitive way. The mapping is however straightforward which leads to the repetition of a cinematic sequence with two identical patterns.

A number of approaches rely on algorithms with restricted camera placement capacities (such as [Blinn 1988]) to position and move the camera in the environment. Recent contributions have proposed techniques to automatically perform these tasks while accounting for more cinematographic properties such as distance to targets, object orientation or visibility. Lino and Christie proposed a toric surface model that efficiently generates a range of viewpoints corresponding to the exact on-screen composition of two or three targets [Lino and Christie 2012]. The technique has been reused by [Galvane et al. 2013] to create dedicated steering behaviors for autonomous cameras and has partly inspired the smooth camera animation system proposed in this paper.

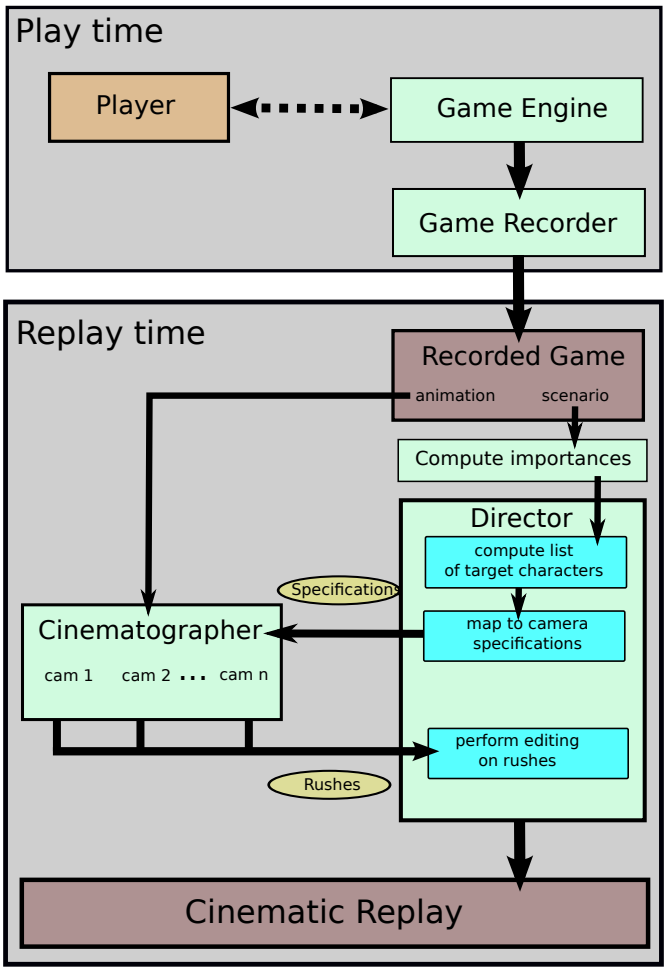


Figure 2: *System overview*

3 Overview

In this section, we give an overview of the method used to generate the cinematic replay from the extraction of the information to the generation of the camera rushes. In order to produce a cinematic replay, we need to access to all the information generated by the game engine. In our case, we devised a game recorder with two purposes: record the animation of the scene (characters, objects, etc.) and retrieve the scenario from the game engine.

Figure 2 shows the different stages of our system. Our system makes use of the recorder with two other components: a director and a cinematographer. Our director's goal is to extract the important targets using the narrative importance of the characters (see Section 4, and then assign camera specifications to the cinematographer using the prose storyboard language (PSL) introduced by [Ronfard et al. 2013] (see Section 5).

It is then the task of the cinematographer to place and move different cameras in the scene. The cinematographer transforms the high-level PSL specifications given by the director into 3D coordinates, and angles for the camera using the geometric information on the scene (see Section 6).

Finally, once all the cameras have been properly animated for the whole replay, the cinematographer sends back to the director the rushes filmed by the cameras. The director is then in charge of performing the editing and creating the final cinematic replay as presented in our results and companion video (see Section 7).

4 An importance-driven approach

In our context, we assume that the game engine is based on a classical two-level narrative structure: beats that describe a narrative unit [Mateas and Stern 2002], and atomic actions that compose a beat (*e.g.* characters moving, speaking or reacting). This is not a general hypothesis nor a prerequisite, and typically can be adapted to whatever information is provided by the game engines.

Using this two-level structure, we compute two different levels of importance. The first level of importance $I_{beat}(c,t)$ provides an estimation of the importance of character c over a specified beat, measured in terms of how many actions he was involved in over the beat occurring at time t and the significance of his role in each action (significance is a numerical value mapping the range "absent" to "significant" to 0..1). The second level, $I_{atomic}(c,t)$ provides an estimation of the importance of a character c from the relevance of the action he's involved in, and from the significance of his role in the action.

$$I_{beat}(c,t) = \sum_{a \in \mathcal{A}_{beat,c}} S_c(a,t)$$

$$I_{atomic}(c,t) = \sum_{a \in \mathcal{A}_{c,t}} R(a,t) \times S_c(a,t)$$

where

- $\mathcal{A}_{beat,c}$ is the set of actions performed by character c in the specified beat;
- $\mathcal{A}_{c,t}$ is the set of actions performed by character c at time t (a character can perform multiple actions simultaneously);
- $R(a,t)$ is the relevance of the action a performed at time t with regards to the completion of the overall objective of the game;
- $S_c(a,t)$ is the significance the role played by the character c in action a at time t.

Values of $R(a,t)$ and $S_c(a,t)$ are considered to be provided by the game engine. While other game engine may provide different information, the key here is to propose a mapping from this information to importances.

5 The Director: from importance to specification of camera behaviors

Once the importances have been computed for each character, we propose to map this list of importances with a collection of camera behaviors. The purpose of this collection is to simultaneously control multiple individual cameras to offer simultaneous distinct viewpoints over which film editing is performed.

5.1 High level specifications

In order to specify camera behaviors, a shot description language is necessary. It defines the specification that will be associated with the characters' importances.

The *Prose Storyboard Language* (PSL) elaborated by [Ronfard et al. 2013] defines the syntax and semantics of a high-level shot description language. Using PSL, partial or full camera specifications can be authored (*i.e* expecting only one solution or a set of possible solutions). Figure 3 shows the subset of the language we focus on. We extended the grammar to handle Point Of View (*POV*) shots (a shot from the physical point of view of the character).

⟨*Composition*⟩ ::= [⟨*angle* ⟩|⟨*pov*⟩] ⟨*FlatComposition*⟩+

⟨*FlatComposition* ⟩ ::= ⟨*size* ⟩ on ⟨*Subject* ⟩[⟨*profile* ⟩] [⟨*screen* ⟩] (and ⟨*Subject* ⟩[⟨*profile* ⟩] [⟨*screen* ⟩] [in (back | fore) ground]) *

⟨*Subject*⟩ ::= (⟨*Actor*⟩|⟨*Object*⟩)+

⟨*angle*⟩ ::= (high | low) angle

⟨*size* ⟩ ::= ECU|BCU|CU|MCU|MS|MLS|FS|LS|ELS

⟨*pov*⟩ ::= POV (⟨*Actor*⟩|⟨*Object*⟩)

⟨*profile* ⟩ ::= 3/4 left back | left | 3/4 left | front | 3/4 right | right | 3/4 left back | back

⟨*screen*⟩ ::= screen (center | left | right)

Figure 3: *Simplified PSL Grammar - Shot specification*

5.2 Behaviors

In order to ease the mapping between the characters' importances and the camera behaviors, we propose to abstract the characters as either PC (player character), P_i (primary non-player characters), and S_i (secondary non-player character). PC is directly provided by the game engine, while P_i and S_i are derived from the relative importance of the characters. We manually determined two threshold values α_S and α_P. At each frame, all the characters (but the player character) with an importance higher than α_P are considered primary characters. The remaining ones with an importance higher than α_S are considered secondary characters. All the others are neglected.

This abstraction creates a configuration of characters for each frame. The different configurations of characters is displayed in Table 1.

Configuration	Meaning
$< PC >$	The player character is the only target
$< P_0 >$	One primary target that is not the player character
$< PC, P_0 >$	Two primary targets, one of which is the player character
$< P_0, P_1 >$	Two primary targets not including the player character
$< P_0, S_0 >$	One primary and one secondary target not including the player character
$< P_+ >$	One primary target or more
$< S_+ >$	One secondary target or more

Table 1: *Different configurations of characters*

A camera behavior is finally constructed by manually mapping a set of PSL shot specifications with a set configurations of characters (one specification per configuration). The configuration represents the stimulus of the camera, and the PSL specification represents the response to the stimulus by the camera. For example, an over-the-shoulder behavior (a point of view always behind the shoulder of the character) can be specified on the player character as illustrated in Table 2. Our system requires a mandatory specification whenever no configuration can be matched (the default case provided in the Table 2).

Behaviors can then be attached to as many cameras as desired in the scene. It is then the role of the Director to select the appropriate rushes from the different cameras.

Configuration	**Specification**
$Default$	MCU on PC 3/4 backright screenleft
$< PC, P_0 >$	CU on PC 3/4 backright screenleft and P_0 screencenter
$< PC, P_+ >$	CU on PC 3/4 backright screenleft and P_+ screencenter

Table 2: *A camera behavior encoding the Over-the-shoulder principle on the player character (PC).*

5.3 Editing

Once the rushes are generated by the different cameras, the director takes care of editing of the rushes to output the final cinematic replay. We perform the editing using the method presented by [Lino et al. 2011a]. This solution consists in casting the problem of film editing as selecting a path inside an editing graph which consists of a collection of segmented rushes. The optimization function relies on a set of costs computed according to basic film grammar rules related to the quality of a shot, the quality of a transition between shots, and the respect of a cutting pace.

6 The Cinematographer: from specifications to camera coordinates

The purpose of the Cinematographer component is to translate a given PSL specification into camera coordinates: position, orientation and focal length for each frame of the animation.

The automated computation of camera coordinates given a PSL specification is not straightforward. The problem is strongly under-constrained – there are many camera coordinates satisfying the same specification – and the space of possibilities is continuous in a 7D space (3D for position, 3D for orientation, 1D for focal length). In related contributions, more general camera description languages have been proposed (see [Olivier et al. 1999; Bares et al. 1998] and [Ranon and Urli 2014]). Sophisticated optimization techniques were proposed to compute camera coordinates by expressing properties of the camera with an aggregated cost function (genetic algorithms in [Olivier et al. 1999], gradient descent [Drucker 1994], particule swarm optimization [Ranon and Urli 2014]).

However, a technique proposed (see [Lino and Christie 2012]) provides an algebraic solution to efficiently solve a subset of camera properties (exact screen location of two targets, distance to targets, viewing angle). We propose to extend this technique for two purposes: (i) to efficiently compute camera coordinates satisfying a PSL specification, and (ii) to smoothly animate the camera while the geometry is changing or to enable smooth transitions between different PSL specifications.

6.1 Computing camera coordinates

We propose to express the computation of camera coordinates from a PSL specification using an incremental pruning process that will successively prune regions of the search space (a spherical surface or a toric surface [Lino and Christie 2012] depending on whether one or two targets are specified, see Figure **??**). The incremental pruning is performed until all properties of the PSL specification are satisfied, or until an inconsistency is encountered. The output of the process is a region of a surface in which all points satisfy the PSL specification.

Given the desired on-screen location of two 3D points A and B, [Lino and Christie 2012] showed that all camera solutions are on a toric surface defined by an arc circle going from A to B, rotated around the segment (AB). This arc circle is displayed in Figure 5. The toric surface is a 2D parametric surface on which, for each point (*i.e.* each camera on the surface), the exact on-screen location of points A and B is satisfied.

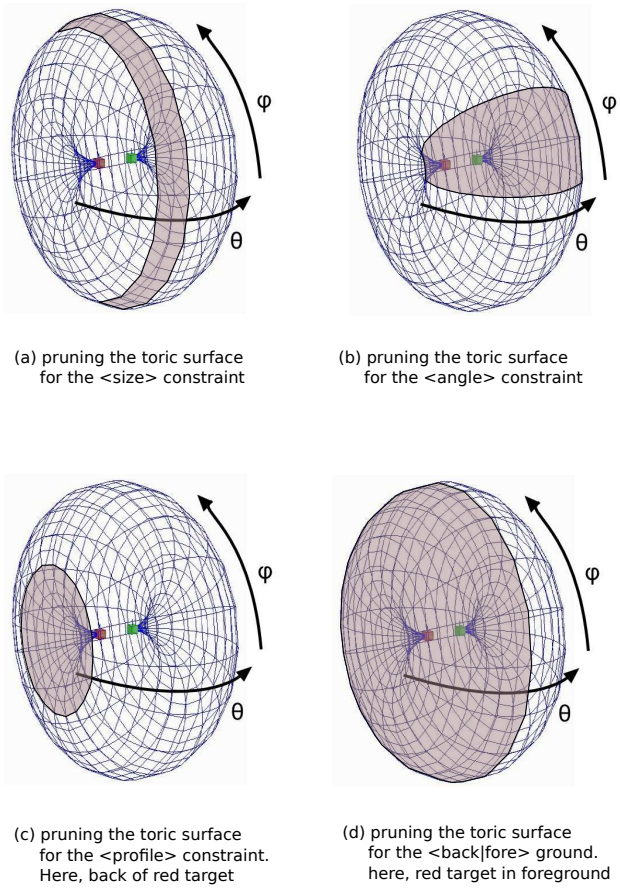
However, when the camera is too close to points A and B on the toric surface, and when considering that A and B are complex objects such as virtual characters, the corresponding viewpoint will be of bad quality (having a camera either inside one of the objects, or too close to an object to create a good viewpoint).

We propose in a first stage to extend this toric surface by introducing a threshold value d preventing the camera from being too close to targets A or B. This occurs at the cost of loosing the exact composition of A and B in these regions, but improves the quality of the viewpoint. The proposition shows to be simpler than our previously proposed extension that relied on Bezier curves [Galvane et al. 2013].

We use this threshold value d between the camera and the targets to alter the surface of the manifold in the following way. For a given arc-circle of the toric (*i.e.* a given value of the vertical angle φ on the parametric surface of the toric), we compute the intersection point I between the arc-circle and the circle defined by either A or B and radius d. The curve is then replaced by the arc circle of center C_2 and radius $|C_2I|$ (in blue on Figure 5) where C_2 is the intersection between (C_1A) and (AB). The arc circle provides a C_1 continuity with the initial curve, hence creating a smooth transition that will be useful when animating the camera (see Section 6.2).

For a PSL specification with two targets (actors or objects), we then apply an incremental pruning process on the extended toric surface by considering the following stages:

1. construct the extended toric surface for the two targets defined by $\langle FlatComposition \rangle$. If no $\langle screen \rangle$ specification is provided, a default one is proposed (first target on the left, second



(a) pruning the toric surface for the <size> constraint

(b) pruning the toric surface for the <angle> constraint

(c) pruning the toric surface for the <profile> constraint. Here, back of red target

(d) pruning the toric surface for the <back|fore> ground. here, red target in foreground

Figure 4: *The pruning process applied on the toric surface to satisfy terms of the PSL shot specification for two targets A and B (displayed in red and green). The intersection of the regions represent the locations in which to position the camera.*

on the right),

2. use the $\langle size \rangle$ specification to compute a vertical band on the toric surface (*i.e.* pruning values of θ). The band represents a range of distances corresponding to the specified shot size. Knowing the camera focal length and the size of the target, the Medium-closeup specification is straightforwardly expressed as a distance δ to the target to which we add some flexibility ($\pm\epsilon$), then converted to a range values for θ – see [Lino and Christie 2012] for more details and illustration in Figure 4(a).

3. use the $\langle angle \rangle$ specification to compute a horizontal band on the toric surface (*i.e.* pruning values of φ see Figure 4(b)),

4. use the $\langle profile \rangle$ specification on each target to prune values of θ, by computing the intersection between the specified wedge of the target (*e.g.* 3/4 left back) and the toric surface (see Figure 4(c))

5. finally use the *(back|fore) ground* specification to decide whether the camera is closer to actor A or to actor B (hence pruning values of θ, see Figure 4(d)).

Each pruning process is performed on the result of the previous stage. Given that some specifications may be optional, not all the pruning stages are performed. At each stage, the pruning process may lead to an empty set corresponding to an inconsistent specification.

For a PSL specification with only one target, the same pruning process is applied on a spherical surface, using spherical coordinates φ, θ, r. In such case, $\langle size \rangle$ defines a range of values for the radius r. For a PSL specification with more than two targets, the toric surface is constructed using the pair of targets having the greatest distance between them.

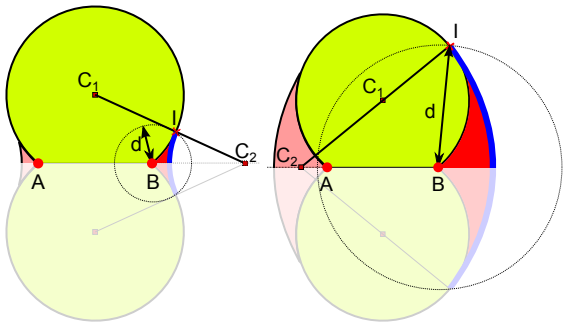


Figure 5: *Modified toric surface. The toric surface is modified for camera positions closer than threshold value d from either target A or B so as to avoid collisions. The process is illustrated for a given value φ of the toric surface, and two different threshold values d. The modified region is replaced by the blue arc circle of center C_2, and radius $|C_2 I|$ where I is the intersection of the circle of center B and radius d, and C_2 is the intersection of lines $(C_1)I$ and (AB).*

Using the spherical or toric surface models, our technique efficiently computes ranges of parameters using the PSL specification. By selecting any given value in these ranges, one can compute a precise camera location and orientation satisfying the specification.

6.2 Animating cameras

The next issue consists in providing means to smoothly animate the camera in two different contexts: (i) maintaining a PSL specification while the scene geometry is evolving, typically when the targets are moving, and (ii) performing transitions between different PSL specifications (either due to a change in the target list, or to a failure in computing a shot satisfying a PSL specification).

Drawing our inspiration from a model we previously developed [Galvane et al. 2013], we propose a physically-based camera animation model that relies on forces directly driven by our extended toric surface or spherical surface and constrained by the 3D environment. The model considers the camera as an oriented particle, influenced by forces guiding both its position and orientation (i) towards the appropriate range of viewpoints satisfying a PSL specification (the positioning force) and (ii) avoiding collisions with obstacles in the environment (the containment force).

The positioning force helps to maintain a consistent framing of the targets by ensuring the continuous satisfaction of a PSL specification. The force is expressed with two distinct forces: one that pushes the camera on the spherical or toric surface, and another force that pushes the camera on the surface until it reaches a desired position. Algorithm 1 details the computation of these two forces. Figure 6 illustrates the idea behind this force using our modified toric surface: we compute the projection P of the camera on the surface and steer the camera to this position (force $\overrightarrow{F_1}$) while pushing the camera on the right hand side or the left hand side, towards

the desired position D (force $\overrightarrow{F_2}$). The camera is steered on the right when the desired position D is on the right side of the vector going from the camera C_i to the point C (middle of the two targets). The camera C_2 illustrates the reason for which we don't simply steer the camera directly towards the desired position: with a camera following the red line, the composition will not be ensured, and the resulting viewpoints would be of low quality (having the camera between the targets).

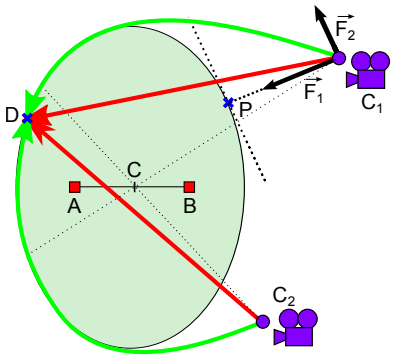


Figure 6: *Steering the camera towards the toric surface (force F_1) and steering the camera along the surface towards target D (force F_2).*

Algorithm 1 Positioning: computes the two forces $F_{projection}$ and $F_{targeting}$ which push the camera towards the desired position while staying on the toric surface. P is the projection of the camera position C_i of camera agent i at time t on the manifold surface and D its desired position. $right$ represents the tangent vector of the manifold surface at P. And v_{max} is the maximum allowed velocity for the camera. v_c represents the current camera velocity.

$F_1 = arrive(P)$
$aim = D - P$
// move the camera to the left or to the right
if desired position on the right **then**
  $dir = right$ *// compute a desired velocity to the left*
else
  $dir = -right$ *// compute a desired velocity to the right*
end if
$u = v_{max}((aim \cdot dir)dir + (aim \cdot up)up)$
// subtract the current velocity to the desired velocity
$F_2 = u - v_c$
$F_{framing} = F_1 + F_2$

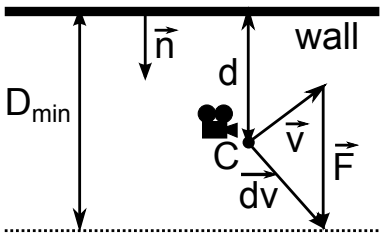


Figure 7: *Obstacle avoidance: compute the force F that pushes the camera C away from an obstacle. D_{min} represents the threshold distance, n is the normal of the surface at the closest distance from the camera to the obstacle, v_c is the velocity of the camera and dv the desired velocity*

The containment force maintains the camera away from obstacles in the environment (typically walls). Figure 7 illustrates the computation of the force and algorithm 2 details its implementation.

Algorithm 2 Containment: computes a sum of forces F_{obs} that pushes the camera away from the obstacles. l_i represents the normalized look at vector (orientation) of camera particule i at time t, r_i represents the normalized right vector of camera particule i at time t and v_{max} is the maximum allowed velocity for the camera. D_{min} represents the distance threshold under which the force is applied.

for each obstacle o **do**
  $d = distanceToObstacle(o, p_i)$
  // check whether the wall is under a threshold distance
  if $d < D_{min}$ **then**
    // compute the magnitude of the force
    $mag = D_{min} - (d + (v \cdot n))$
    $F_{obs} = F_{obs} + n * mag$
  end if
end for

The key benefit of this physical camera animation system is to generate smooth camera motions, and to provide control over the transitions between different PSL specifications.

6.3 Filtering

Using a physically based model to control cameras offers a practical way to avoid unrealistic camera movements and ensures continuity. The current solution however comes with a drawback: since the toric and spherical surfaces are directly computed from targets' positions, any noisy motions in these positions (nodding, head motion due to walking) will directly impact the camera motions. Figure 8 illustrates these issues on target trajectories. Even though the use of a physical system can dampen some of these noisy motions, a more elaborate model is necessary.

(a) *Continuous head movement* **(b)** *Oscillations of the character's head*

Figure 8: *A denoising algorithm is applied on the motion of the character (eg balancing head motions, or head walking motions) to prevent noisy camera trajectories.*

While a simple solution could be to apply thresholds to the camera forces to prevent moving the camera when unnecessary, it requires a lot of parameter tuning, induces undesirable motion such as peaks in the acceleration and leads to latency in camera tracking.

To solve this problem, we cast it into a denoising problem by considering the small variations in the target trajectories as noise. The filtered trajectories are then obtained using a total variation (*TV*) regularization algorithm. The idea of using *TV* algorithm for denoising was introduced by [Rudin et al. 1992]. The idea behind the *TV* denoising problem is the following: we are given a (noisy) signal $y = (y[1], ..., y[N]) \in \mathbb{R}^N$ of size $N \geq 1$, and we want to efficiently compute the denoised signal $x* \in \mathbb{R}^N$, defined implicitly as the solution to the following minimization problem with a regularization parameter $\lambda \geq 0$:

$$\underset{x\in\mathbb{R}^N}{minimize}\frac{1}{2}\sum_{k=1}^{N}\left|y[k]-x[k]\right|^2+\lambda\sum_{k=1}^{N-1}\left|x[k+1]-x[k]\right|$$

For the purpose of filtering trajectories, the denoising is performed by applying the *TV* regularization to each of the coordinates (*x*, *y* and *z*) over time (*N* thus represents the number of frames of the sequence). To obtain smooth and steady camera movements, we propose to denoise the target's trajectories as a pre-process (rather than denoise the computed camera motions). We keep the advantage of the force-based system by tracking trajectories that have already been filtered and thus do not induce extra forces to constantly adjust the camera when it is not needed.

For denoising the trajectories, we used a direct non-iterative algorithm presented by [Condat 2013]. Finding the appropriate value for parameter λ was performed through multiple experimentations. The value was finally set to *2.0*.

7 Experimental results

To demonstrate our approach, we used the video game *Nothing For Dinner*. This interactive drama presented in [Habonneau et al. 2012] and available online[2] uses the story engine IDtension. The goal of this serious game is to help teenagers cope when a parent suffers from traumatic brain injury. The simulation immerses the players in an interactive environment in which they play active roles and have to make decisions that require their attention. The gaming experience provided by *Nothing For Dinner* gives users a way to experience different situations that they might encounter in their everyday life. We integrated our cinematic replay system within this serious game, giving the possibility for users to replay their experiences.

7.1 Narrative importance

All the narrative information is generated by the IDtension engine and saved for further analysis by our system. What is being generated by IDtension could be considered as the fabula of the story: it contains all events and actions occurring during the game session along with their temporal relations within the fictional world without any consideration of viewpoint or focalisation. To generate the cinematic replay, our system extracts information from this fabula, typically beats and atomic actions. Each atomic action is described with the following attributes: *starting time*, *duration*, *type of actions* and *description*.

The information on the relevance of the actions performed by the characters is part of the internal mechanisms of *IDTension*. It is termed *motivation* and corresponds to the relevance of the action in terms of the accomplishment of the character's goal. Combined with the significance of character's role in each action, this metric provides a means to establish the individual importances of the characters.

7.2 Shots specifications

For the results, we demonstrate the capacities of our system by using only 3 cameras. Two cameras rely on the fine-grain importance I_atomic and the third one (the master shot) relies on the beat importance I_{beat}. Tables 3, 4 and 5 describe the behaviors defined for each of these cameras. With this implementation, the first camera represents the Point-Of-View shot from the player character's perspective and the second camera represents its reverse shot.

Configuration	Specification
$Default$	MS on PC right screenleft
$< P_0 >$	POV PC on P_0 screencenter
$< PC, P_0 >$	POV PC on P_0 screencenter
$< PC, P_+ >$	POV PC on P_+ screencenter

Table 3: *Behavior for the first camera*

Configuration	Specification
$Default$	MCU on PC 3/4 backright screencenter
$< P_0 >$	CU on PC 3/4 right screencenter
$< PC, P_0 >$	CU on PC 3/4 right screencenter
$< PC, P_+ >$	CU on PC 3/4 right screencenter

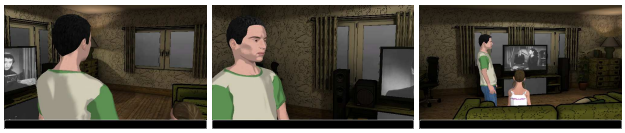
Table 4: *Behavior for the second camera*

Configuration	Specification
$Default$	MCU on PC right screenleft
$< P_0 >$	MS on PC screenleft, P_0
$< PC, P_0 >$	MS on PC screenleft, P_0
$< PC, P_+ >$	MS on PC screenleft, P_+

Table 5: *Behavior corresponding to a master shot*

7.3 Computing camera positions

Evaluating cinematography is always a delicate matter. We present qualitative results produced by our system. Figures 9, 10 and 11 show shots generated for different situations using different camera behaviors. Figure 9 shows the output of the three cameras when no specific action is occurring. The camera simply performs a tracking of the player character PC. Figure 10 shows the results obtained in a situation of dialog between the player character and another character. Figure 10a shows the Point Of View shot obtained using the set of rules previously defined. And Figure 10b shows its reverse shot: the internal shot.



(a) *First camera* **(b)** *Second camera* **(c)** *Master shot*

Figure 9: *Shots computed for three different camera behaviors on the same scene at the same time (a) first camera behavior, (b) second camera behavior and (c) master shot behavior.*

To illustrate the benefit of the system, we show how a change in camera behaviors impacts the computed viewpoints. Rather than using a Point Of View shot combined with an internal shot, we used two complementary Over-The Shoulder-shots. To produce the result displayed in Figure 11 – to be compared with Figure 10, we simply replaced the following rules respectively for the first and second cameras, thereby offering simple means for users to author their cinematic replays without manually moving the camera and re-editing the sequence.

- $< PC, P_0 >$: CU on PC 3/4backright screenleft and P_0 center

[2] http://tecfalabs.unige.ch/tbisim/portal/

(a) *First camera* **(b)** *Second camera* **(c)** *Master shot*

Figure 10: *Shots computed for three different camera behaviors on the same scene at the same time: (a) point-of-view behavior defined on the PC Frank, (b) point-of-view behavior defined on P_0 Lili and (c) master shot behavior defined on $< PC, P_0 >$.*

- $< PC, P_0 >$: CU on P_0 3/4backleft screenright and PC center

(a) *First camera* **(b)** *Second camera* **(c)** *Third camera*

Figure 11: *Shots computed for three different camera behaviors on the same scene at the same time: (a) over-the-shoulder behavior defined on Frank, (b) over-the-shoulder behavior defined on Lili and (c) master shot behavior on Franck and Lili.*

7.4 Overall process and results

These few examples illustrate the type of camera shots generated by our system. It highlights the complementarity of the behaviors in generating various shots that makes the editing process easier. To illustrate the overall system, Figure 12 presents the complete cinematic replay process. The process starts with the analysis of the list of actions and activities to compute both the atomic and beat importances of the character along the time-line. The figure shows the evolution of the atomic importance of the characters over time (the same computation is performed for the beat importance). Using this information at each time step we can extract the list of characters involved (configurations of characters) and use it to define the camera specifications from the set of behaviors presented in Tables 9, 10 and 11 (in this case, Frank is the Player Character). The rushes are then computed from the camera specifications using the steering behaviors and the editing between the different rushes is performed.

Finally, the companion video presents two different replays of the same game session. They were obtained by changing the behaviors of the camera as mentioned before. This video shows that a small set with three cameras and only a few rules is enough to cover basic interactions between characters and transitions between actions.

8 Limitations and future work

The focus on this paper was set on the generation of cinematic replays for dialogue-based role playing games. It provides a generic solution for this purpose but doesn't make full use of the narrative information that some games or interactive narratives might provide. Looking at richer information, the proposed cinematography system could be improved, for example by addressing the emotion of the characters. Though the game *Nothing For Dinner* itself provides us with such information, the automated computation of compelling cinematographic sequences conveying emotions remains an open challenge.

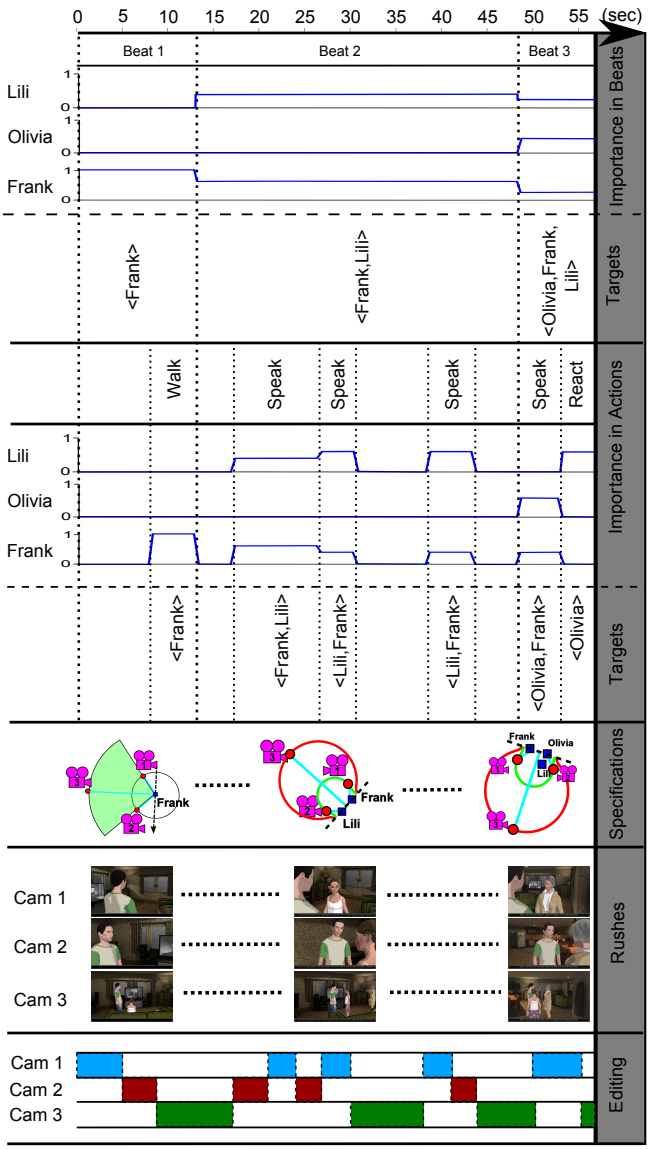


Figure 12: *An illustration of the complete cinematic replay process. Starting with the list of beat and actions, the importances of the characters are computed to establish the configurations of characters. Each configuration is converted into camera specifications given each of the camera behaviors (3 behaviors are defined here). Finally, the rushes are generated from the specifications and edited.*

Finally, for this research, we focused on the cinematic replay aspect and made use of the off-line advantages of such goal. Nevertheless, our cinematography system could be adapted to address real-time contexts. This however requires the provision of an on-line camera editing technique which presents complex issues of it's own due to the impossibility of knowing the evolution of the scene in advance. Nevertheless, the provision of automated cinematography systems to real-time applications remains our objective, in a way to enhance the player's experience in the gameplay itself.

9 Conclusion

In this paper we have presented a new system designed to automatically generate cinematic replays of game sessions. We presented a new way to define high-level camera specifications using a principled and contextual importance-driven approach, as an answer to the limitations of action-based or idioms-based cinematography. We also introduced a mean to express camera behaviors using these specifications, and proposed novel techniques to smoothly animate the cameras. The results obtained with only three camera behaviors illustrate the capacity of the system to properly convey a game replay, with a realistic and dynamic camera style.

Acknowledgements

This work has been funded by the French ANR Chrome and ANR Cinecitta research projects.

References

BARES, W. H., GREGOIRE, J. P., AND LESTER, J. C. 1998. Realtime Constraint-Based cinematography for complex interactive 3D worlds. In *Proceedings of AAAI-98/IAAI-98*, 1101–1106.

BLINN, J. 1988. Where am I? what am I looking at? *IEEE Computer Graphics and Applications 8*, 4 (July), 76–81.

CHEONG, Y.-G., JHALA, A., BAE, B.-C., AND YOUNG, R. M. 2008. Automatically generating summary visualizations from game logs. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, The AAAI Press.

CHRISTIANSON, D. B., ANDERSON, S. E., HE, L.-W., WELD, D. S., COHEN, M. F., AND SALESIN, D. H. 1996. Declarative camera control for automatic cinematography. In *Proceedings of AAAI '96*, 148–155.

CONDAT, L. 2013. A Direct Algorithm for 1D Total Variation Denoising. *IEEE Signal Processing Letters 20*, 11, pp. 1054 – 1057.

DELOURA, M. 2009. *Real Time Cameras, A Guide for Game Designers and Developers*. Morgan Kaufman.

DOMINGUEZ, M., YOUNG, R. M., AND ROLLER, S. 2011. Automatic identification and generation of highlight cinematics for 3d games. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, ACM, 259–261.

DOMINGUEZ, M., YOUNG, R. M., AND ROLLER, S. 2011. Design and evaluation of afterthought, a system that automatically creates highlight cinematics for 3d games. In *AIIDE*, The AAAI Press.

DRUCKER, S. M. 1994. *Intelligent Camera Control for Graphical Environments*. PhD thesis, School of Architecture and Planning, Massachusetts Institute of Technology MIT Media Lab.

ELSON, D. K., AND RIEDL, M. O. 2007. A lightweight intelligent virtual cinematography system for machinima generation. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE '07)*.

GALVANE, Q., CHRISTIE, M., RONFARD, R., LIM, C.-K., AND CANI, M.-P. 2013. Steering behaviors for autonomous cameras. In *Proceedings of Motion on Games*, ACM, MIG '13, 71:93–71:102.

HABONNEAU, N., SZILAS, N., RICHLE, U., AND DUMAS, J. 2012. 3D Simulated Interactive Drama for Teenagers coping with a Traumatic Brain Injury in a Parent. In *5th International Conference on International Digital Storytelling (ICIDS 2012). LNCS 7648*, Springer, Heidelberg, D. Oyarzun, F. Peinado, R. M. Young, A. Elizalde, and G. Méndez, Eds., 174–182.

HALPER, N., HELBING, R., AND STROTHOTTE, T. 2001. A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. *Computer Graphics Forum 20*, 3, 174–183.

HAWKINS, B. 2005. *Real-Time Cinematography for Games*. Charles River Media.

HE, L.-W., COHEN, M. F., AND SALESIN, D. H. 1996. The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In *ACM SIGGRAPH*, SIGGRAPH '96, 217–224.

JHALA, A. 2006. Darshak: an intelligent cinematic camera planning system. In *AAAI'06: proceedings of the 21st national conference on Artificial intelligence*, AAAI Press, AAAI Press, 1918–1919.

LINO, C., AND CHRISTIE, M. 2012. Efficient Composition for Virtual Camera Control. In *ACM Siggraph / Eurographics Symposium on Computer Animation*, P. Kry and J. Lee, Eds.

LINO, C., CHOLLET, M., CHRISTIE, M., AND RONFARD, R. 2011. Computational Model of Film Editing for Interactive Storytelling. In *ICIDS 2011 - International Conference on Interactive Digital Storytelling*, Springer, Vancouver, Canada, 305–308.

LINO, C., CHRISTIE, M., RANON, R., AND BARES, W. 2011. The director's lens: An intelligent assistant for virtual cinematography. In *Proceedings of the 19th ACM International Conference on Multimedia*, ACM, New York, NY, USA, MM '11, 323–332.

MARKOWITZ, D., KIDER, J. T., SHOULSON, A., AND BADLER, N. I. 2011. Intelligent camera control using behavior trees. In *Proceedings of the 4th International Conference on Motion in Games*, Springer-Verlag, Berlin, Heidelberg, MIG'11, 156–167.

MATEAS, M., AND STERN, A. 2002. A behavior language for story-based believable agents. *IEEE Intelligent Systems 17*, 4 (July), 39–47.

OLIVIER, P., HALPER, N., PICKERING, J., AND LUNA, P. 1999. Visual Composition as Optimisation. In *AISB Symposium on AI and Creativity in Entertainment and Visual Art*, 22–30.

RANON, R., AND URLI, T. 2014. Improving the efficiency of viewpoint composition. *IEEE Trans. Vis. Comput. Graph. 20*, 5, 795–807.

RONFARD, R., GANDHI, V., AND BOIRON, L. 2013. The Prose Storyboard Language: A Tool for Annotating and Directing Movies. In *2nd Workshop on Intelligent Cinematography and Editing part of Foundations of Digital Games - FDG 2013*.

RUDIN, L. I., OSHER, S., AND FATEMI, E. 1992. Nonlinear total variation based noise removal algorithms. *Phys. D 60*, 1-4 (Nov.), 259–268.

TRUFFAUT, F., AND SCOTT, H. G. 1967. *Hitchcock*. Simon & Schuster.